CSN08x14

Scripting for Cybersecurity and Networks

Lecture 6: Probability and Statistics; Tuning Python code

Today's Topics

You will learn about:

- Sequences, Permutations and Combinations
- Binomial Coefficients
- What are the chances?
- Probability Theory - very basic intro
- Basic statistics
- Code profiling / Tuning Python code

Go to www.menti.com

code **xxx**

Sampling (selection) methods

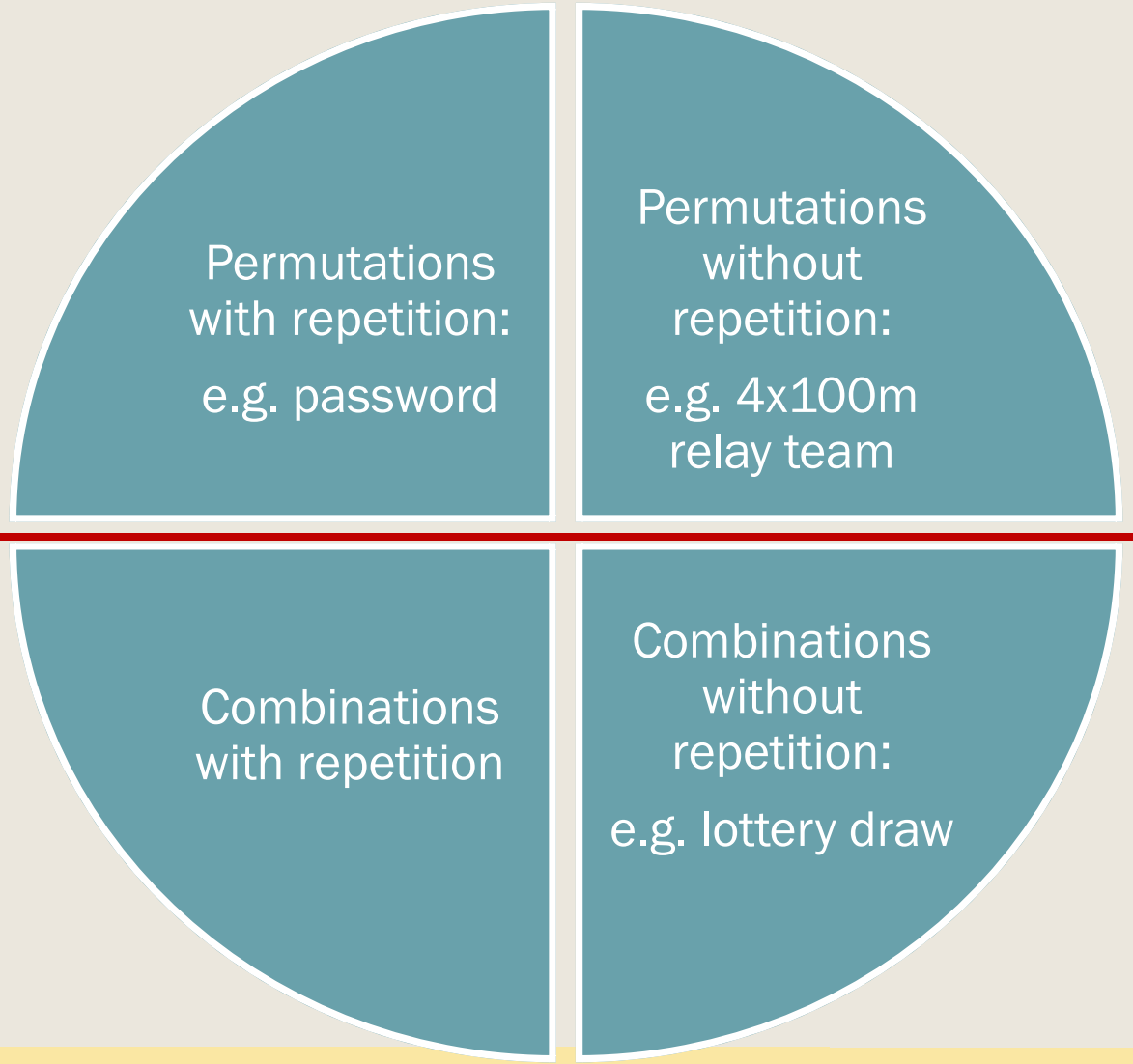
Permutation: order is important

Combination:

order does not matter

With repetition: items can be used multiple times

Without repetition: each item can be used only once



Permutations 1: Sampling (selection) with repetition, where order is important

Which bike lock is better?

- *Model A: 4 rings with digits 0-9*
- *Model B: 5 rings with digits 0-6*

How many possibilities?

- How many different sequences are possible?

- E.g. How many possible values for a 4-digit PIN?
  - *10000 possibilities... 0000, 0001, 0002, ... 9998, 9999*
  - *In each of the four positions there are 10 options so the result is 10x10x10x10 or 10^4*

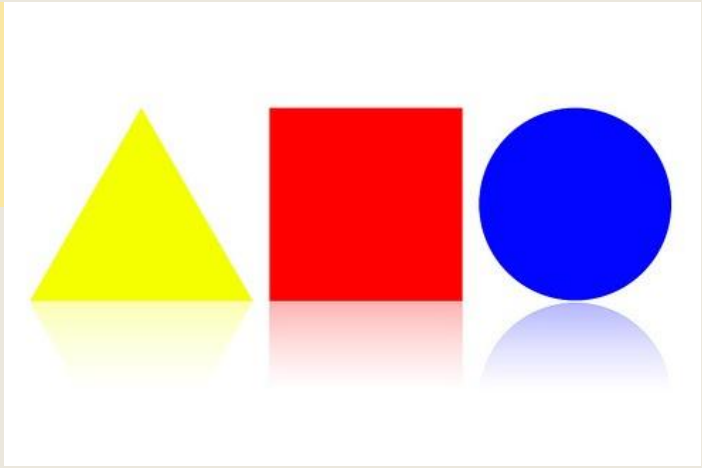
Probability

- The probability of a specific event occurring is

  P = 1 / (number of possible events)

- You flip one coin and roll one die. What is the probability of getting heads and 6?

P( and ) = ?



- We have different shapes which include “small red triangle”, “big blue circle” etc - all of the permutations of (small|big) (red|green|blue) (circle|square|triangle)
- Let's say we know that 4b23449b89f1322cbcdc8620d1672f28 is the md5 hex digest of one of them.
- How many possibilities are there?
- What are the chances of our first guess being correct?

Solving the problem in python (permutations.py)

permutations.py - F:/Dropbox/CSN08114 Python/permutations.py (3.7.0)

File Edit Format Run Options Window Help

```
import hashlib

def allPerms(A,B,C):
    z=[(a,b,c) for a in A for b in B for c in C]
    yield z

for triple in allPerms(["big","small"],
                       ["red","green","blue"],
                       ["circle","triangle","square"]):
  for i in triple:
    phrase= f'{i[0]} {i[1]} {i[2]}'
    m=hashlib.md5(phrase.encode('utf-8')).hexdigest()
    if m.startswith('4b23'):
      print(phrase)
```

Ln: 15 Col: 0

Permutations in Python

- Memory intensive if using lists of tuples
- Generator creates lists on the fly
  - *Use "yield" instead of "return"*
  - *See https://www.pythoncentral.io/python-generators-and-yield-keyword/*

Example (permutations1.py)

```
def allPerm_gencomp(A,B,C):
    """Generator for all possible permutations of A,B,C from list comprehension"""
    z=[(a,b,c) for a in A for b in B for c in C]
    yield z

def allPerm_comp(A,B,C):
    """List of all possible permutations of A,B,C from list compre
    z=[(a,b,c) for a in A for b in B for c in C]
    return z

def main():
    A,B,C = ["big","small"],["red","green","blue"],["circle","triangle","square"]
    y = allPerm_comp(A,B,C)
    print(f'**USING RETURN**\ny = {y}\n')
    print(f'len(y) = {len(y)}\n')
    x = allPerm_gencomp(A,B,C)
    print(f'**USING YIELD**\nx = {x}\nx = ', end='')
    for triple in x: print(triple)
    print(f'len(x) = {len(x)}')

if __name__ == '__main__':
    main()
```

Apart from yield/return, both functions are identical

Example (permutations1.py)

```
============ RESTART: F:/Dropbox/CSN08114 Python/permutations1.py ============
**USING RETURN**
y = [('big', 'red', 'circle'), ('big', 'red', 'triangle'), ('big', 'red', 'square'), ('big', 'green', 'circle'), ('b
ig', 'green', 'triangle'), ('big', 'green', 'square'), ('big', 'blue', 'circle'), ('big', 'blue', 'triangle'), ('big
', 'blue', 'square'), ('small', 'red', 'circle'), ('small', 'red', 'triangle'), ('small', 'red', 'square'), ('small'
, 'green', 'circle'), ('small', 'green', 'triangle'), ('small', 'green', 'square'), ('small', 'blue', 'circle'), ('s
mall', 'blue', 'triangle'), ('small', 'blue', 'square')]

len(y) = 18

**USING YIELD**
x = <generator object allPerm_gencomp at 0x0000000002F2EB88>
x = [('big', 'red', 'circle'), ('big', 'red', 'triangle'), ('big', 'red', 'square'), ('big', 'green', 'circle'), ('b
ig', 'green', 'triangle'), ('big', 'green', 'square'), ('big', 'blue', 'circle'), ('big', 'blue', 'triangle'), ('big
', 'blue', 'square'), ('small', 'red', 'circle'), ('small', 'red', 'triangle'), ('small', 'red', 'square'), ('small'
, 'green', 'circle'), ('small', 'green', 'triangle'), ('small', 'green', 'square'), ('small', 'blue', 'circle'), ('s
mall', 'blue', 'triangle'), ('small', 'blue', 'square')]
Traceback (most recent call last):
  File "F:/Dropbox/CSN08114 Python/permutations1.py", line 44, in <module>
    main()
  File "F:/Dropbox/CSN08114 Python/permutations1.py", line 41, in main
    print(f'len(x) = {len(x)}')
TypeError: object of type 'generator' has no len()
```

- Generator has no length
- Need to iterate over it to print/use

Permutations2: Selection/sampling without repetition, where order is important

4x100m relay team

- You have a squad of 6 runners and you must select a crew of 4 – the order matters.
  - *How many permutations are there?*

- The first position can be one of 6 places, the second is one of 5…
- We write ${}_6P_4$ or P(6,4) for the number of permutations of 4 things from a pool of 6
- P(6,4) = 6*5*4*3

Permutations: P(n,r) in general

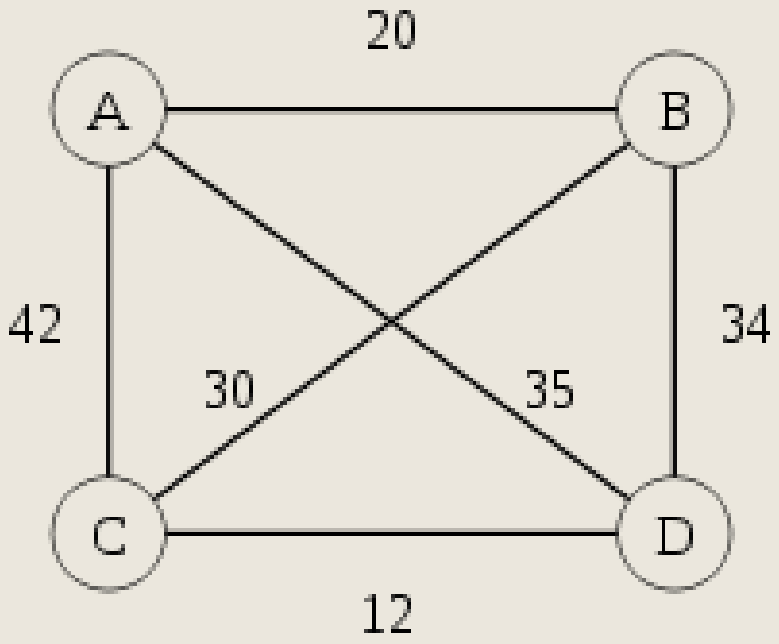
- The number of permutations of n different things is n!
- If you can choose *r* different things from a pool of *n* then the calculation is **n perm r**

$$\mathrm{P(n,r)} = {}_{n}\mathrm{P_r} = \frac{n!}{(n-r)!}$$

Online calculator https://www.calculatorsoup.com/calculators/discretemathematics/permutations.php

Traveling salesman problem

- Find the shortest route that visits every city
- A naive solution requires examination of every one of the *n!* possible paths
  - *ABCD (20+30+12=62),*
  - *ABDC (66),*
  - *ACBD, ACDB,*
  - *BACD…*



Binomial Explosion

- The Travelling Salesman Problem is hard because the number of possibilities rises so quickly.
- The expression *n!* is super-polynomial

- Animation at https://www.youtube.com/watch?v=SC5CX8drAtU

Combinations2: Selection/sampling without repetition, where order does NOT matter

Example: football tournament

- Football tournament with 4 teams ABCD, every team must play every other team once
- Equivalent to choosing 2 letters from set of {A,B,C,D}, the order is not important
- We know the permutations are AB,AC,AD,BA,BC,BD,CA,CB,CD,DA,DB,DC
- But as the order is not important, AB is the same as BA. So we have counted each solution twice
- Calculate ${}_4C_2$:
  ${}_4C_2 = {}_4P_2 / 2 = 4*3/2 = 6$

	Team A	Team B	Team C	Team D
Team A	-	AB	AC	AD
Team B	BA	-	BC	BD
Team C	CA	CB	-	CD
Team D	DA	DB	DC	-

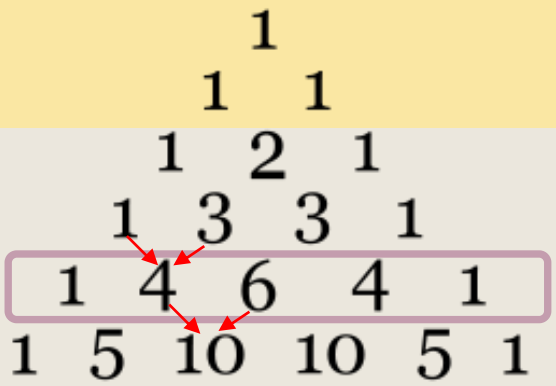
Combinations (without replacement)

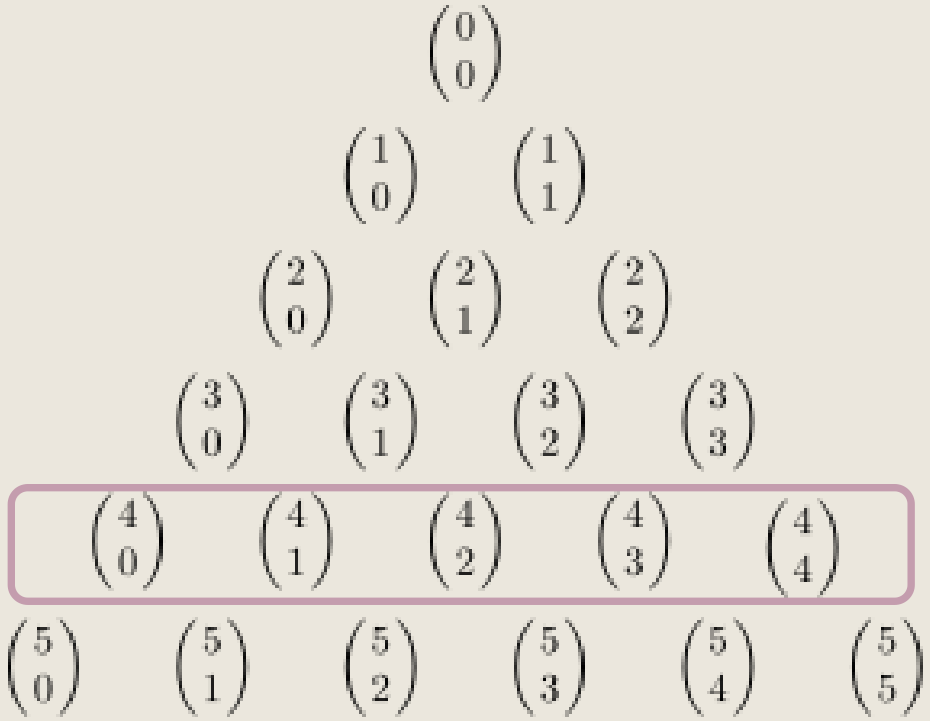
- In combinations we do not care about the order of the result
- Without replacement - Each value in the pool can be used only once
- ${}_nC_r$ is **n choose r** or C(n, r)

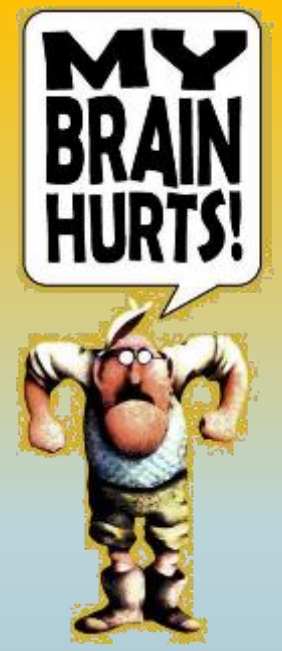
$$ {}_nC_r = C(n,r) = \binom{n}{r} = \frac{n!}{r!(n-r)!} $$

Pascal's triangle

- Very easy to generate - each value is the sum of the two above it
- Actually shows the possible combinations ${}_nC_r$







Probability: Repeated trials

Example: Penalty shoot-out

- Probability of scoring a penalty is 3/4
- Your team gets 5 shots.
- What is the probability that you will
  - *Score 5?*
  - *Score 0?*
  - *Score 4?*

Probability of *r* in *n* independent trials

- The probability of a success (in one trial) is *p*
- The probability of a failure is (in one trial) *(1-p)*

- The probability of exactly *r* successes in *n* trials is

$$ {}_nC_r \, p^r \, (1-p)^{(n-r)} $$

- Assumes trials are independent of each other
- Can use formula for failures as well

Cumulative probability

- Probability of scoring a penalty is 3/4. Your team gets 5 shots.
- What are your chances of scoring **at least 3** goals? (**no more than 2** failures)
- i.e. chances of scoring 3 or 4 or 5 goals from 5 attempts
- We can take p(5 goals)+p(4 goals)+p(3 goals) =p(0 fails)+p(1 fail) + p(2 fails)

attempts:	5			
p(failure):	0.25			
			exact	cumulative
probability that	0	will fail:	0.2373047	0.237305
probability that	1	will fail:	0.3955078	0.632813
probability that	2	will fail:	0.2636719	**0.896484**
probability that	3	will fail:	0.0878906	0.984375
probability that	4	will fail:	0.0146484	0.999023
probability that	5	will fail:	0.0009766	1.000000

Cumulative probability examples

- The crux is always whether the trials really are independent of each other
- Bad blocks on a disk: e.g. you have 100 blocks, probability of a block being a bad one is 1/256. What is the probability of 3 or fewer bad blocks?
- How many spares do you need?
  You are in charge of 1000 machines. Each machine is unreliable: the probability that a machine will fail (in one day) is 1/1000. You can restock spares in one day, so need enough spares for one day. You want the probability of running out to be 1:1,000,000 or less?

How many spares?

- If we have 1000 machines then we need 9 spares to be sure that the chance of running out is less than 1 in 1 million
  (i.e. we need >= 0.999999)
  - *The probability that 9 or fewer of 1000 will fail is 0.9999998926*
- If we have 2000 machines then we need 12 spares to be sure...
  - *The probability that 12 or fewer of 2000 will fail is 0.9999997984*

1000 machines				Cumulative
Probability that	0	will fail	0.367695	0.3676954248
Probability that	1	will fail	0.368063	0.7357589130
Probability that	2	will fail	0.184032	0.9197906572
Probability that	3	will fail	0.061283	0.9810731665
Probability that	4	will fail	0.01529	0.9963631220
Probability that	5	will fail	0.003049	0.9994119299
Probability that	6	will fail	0.000506	0.9999180300
Probability that	7	will fail	7.19E-05	0.9999899681
Probability that	8	will fail	8.94E-06	0.9999989064
Probability that	9	will fail	9.86E-07	**0.9999998926**

2000 machines				Cumulative
Probability that	0	will fail	0.1352	0.1351999254
Probability that	1	will fail	0.270671	0.4058704467
Probability that	2	will fail	0.270806	0.6766764388
Probability that	3	will fail	0.180537	0.8572137668
Probability that	4	will fail	0.090223	0.9474372513
Probability that	5	will fail	0.036053	0.9834905196
Probability that	6	will fail	0.012	0.9954902310
Probability that	7	will fail	0.003422	0.9989118561
Probability that	8	will fail	0.000853	0.9997651218
Probability that	9	will fail	0.000189	0.9999541669
Probability that	10	will fail	3.77E-05	0.9999918435
Probability that	11	will fail	6.82E-06	0.9999986664
Probability that	12	will fail	1.13E-06	**0.9999997984**

Permutations and combinations in Python

- Use the itertools library to generate all possible permutations / combinations, e.g.
  - **itertools.combinations**([1,2,3],2)
  - **itertools.permutations**([1,2,3])
  - itertools.permutations([1,2,3],2)

- See e.g. https://www.geeksforgeeks.org/permutation-and-combination-in-python/

Permutations and combinations in Python

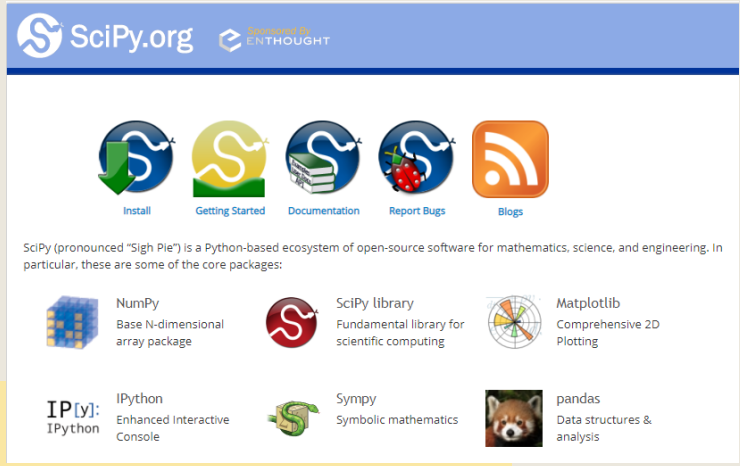
- Using itertools is inefficient to only calculate the number / probability
- Essential ingredients are n!, ${}_nP_r$ and ${}_nC_r$.
- n! is **math.factorial**(n)
- Scipy is good for scientific computing. Implements efficient functions for permutations and combinations

  scipy.special.perm(n,r) $\#_nP_r$

  scipy.special.comb(n,r,repetition=False) $\#_nC_r$

- Scipy needs to be installed (pip)

https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.perm.html
https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.comb.html



PROBABILITY THEORY AND STATISTICS

Probability theory

can help us answer questions that involve uncertainty, such as determining whether:

- we should reject an incoming mail message as spam based on the words that appear in the message
- A virus checker should flag up code as malicious or not

What would we need to calculate a "spam probability" for a new email message text?

- Let's say we have calculated spam probabilities for 5 new emails.

Message	Spam probability	Reject?
1	0.0004	
2	0.36	
3	0.51	
4	0.66	
5	0.986	

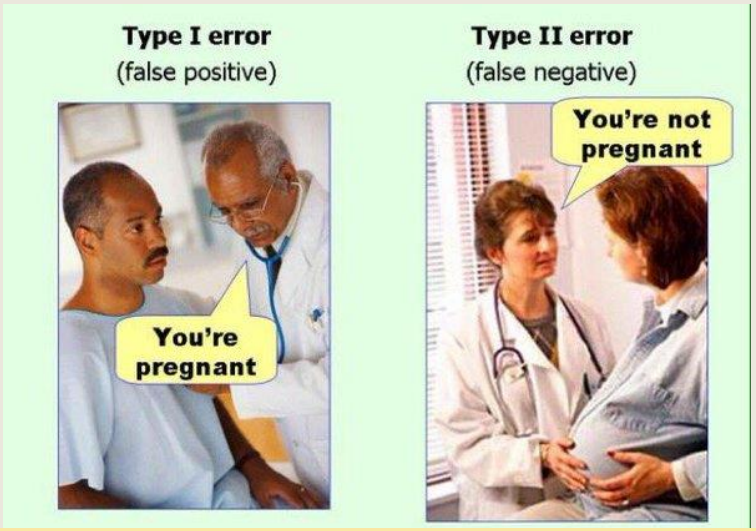
- Which of these would you actually reject (classify as spam)?

Q: What can go wrong?

What can go wrong?

- Type I error (false positive)
- Type II error (false negative)

Really → / Classified as ↓	Email is OK	Email is Spam
OK	✓	✗
Spam	✗	✓



Basic statistics - mean, median and mode

- All measure the "centre" or central tendency of a list of values
  - *Purpose: give a one-number summary for easy comparison*

- Mean - what is usually meant by "average"
  - *Sample mean (x-bar)*

$$\bar{x} = \frac{(x_1 + x_2 + \cdots + x_n)}{n}$$

Population mean (mu)

$$\mu = \frac{\sum x}{n}$$

- Median - the middle value
  - *Arrange all values in order, the median is the middle value. (or average of two middle values is n even)*
- Mode - the most frequent score (used for categorical data)

Mean, median, mode - which is best?

mean	median	mode
• Easy to calculate • Affected by outliers and skewed distributions • Very well-known	• Calculation more complex • Less affected by outliers and skew • Easy to interpret • Less well-known than mean(?)	• Need not be unique • Not suitable for continuous data or where each value occurs only once or twice • For categorical data only the mode can be used

Examples & explanation see e.g.
http://www.abs.gov.au/websitedbs/a3121120.nsf/home/statistical+language+-+measures+of+central+tendency

Some argue that the median is the best of these measures!
(https://learnandteachstatistics.wordpress.com/2013/04/29/median/)

Example

- Sue, Lynn and Mark have 30 chicken nuggets
- The mean (average) is 10 each
- What does this tell us?

Mean=10 chicken nuggets	Example 1	Example 2
Sue 	10 	2 
Lynn 	10 	5 
Mark 	10 	23 
	Median =10	Median =5

Basic statistics: Measures of spread

- How spread out (similar or varied) the data are
- There are various measures of spread. You need to choose one appropriate for the central measure you are using
- See http://www.abs.gov.au/websitedbs/a3121120.nsf/home/statistical+language+-+measures+of+spread

Measure of spread	Goes with..
Variance	mean
Standard deviation (=sqrt(variance))	mean
Quartiles	median
Min & max	median
range	median
Inter-quartile range	median

Variance
(very roughly the average distance from the mean)

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

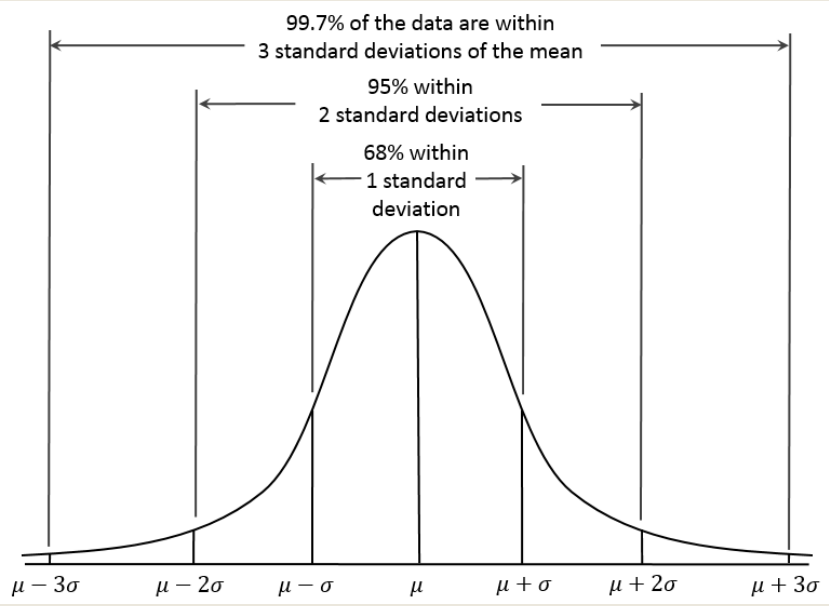
Quartiles split the data into equal quarters
(Q2 = median)
Interquartile range = Q3-Q1

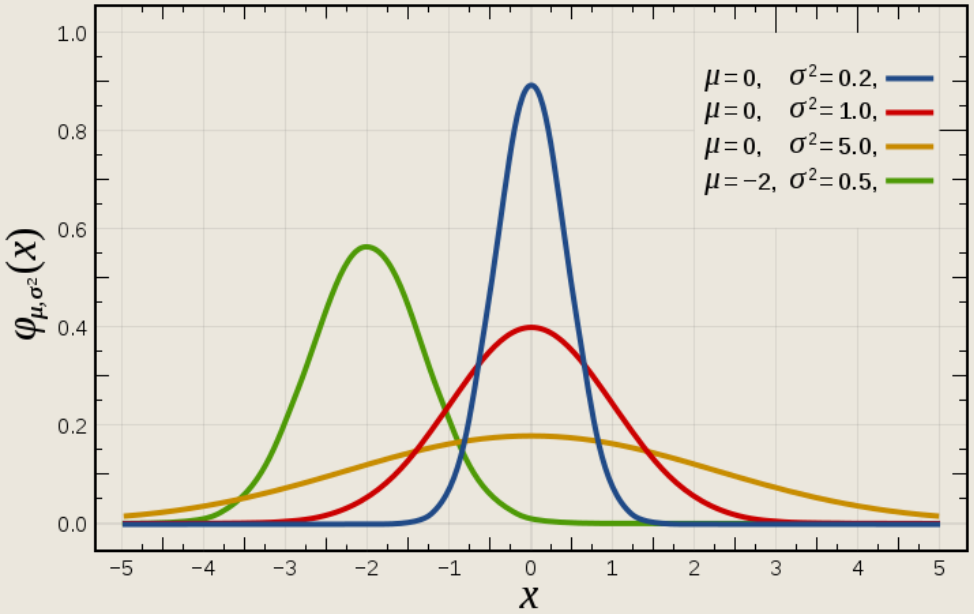
25% of values	Q1	25% of values	Q2	25% of values	Q3	25% of values

Scripting for Cybersec & Networks

Normal distribution and Standard deviation

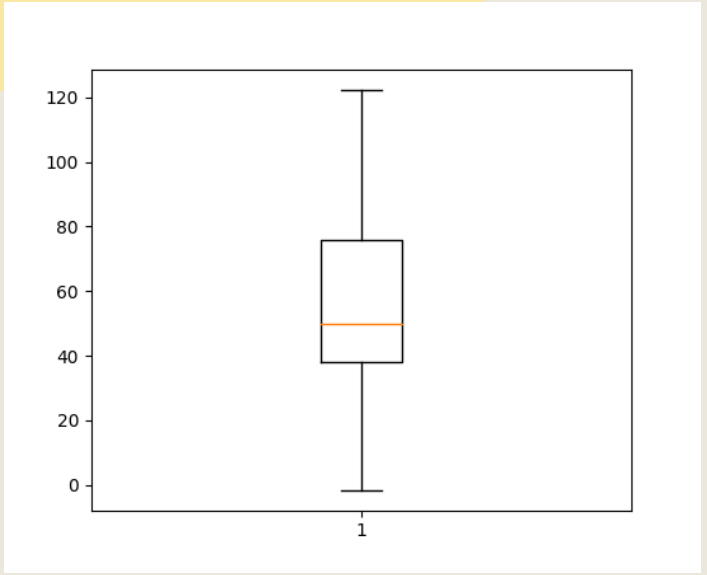
- Normal distribution:
  - 68% of values are within one standard deviation of the mean
  - 95% within two standard deviations
  - 99.7% within three standard deviations



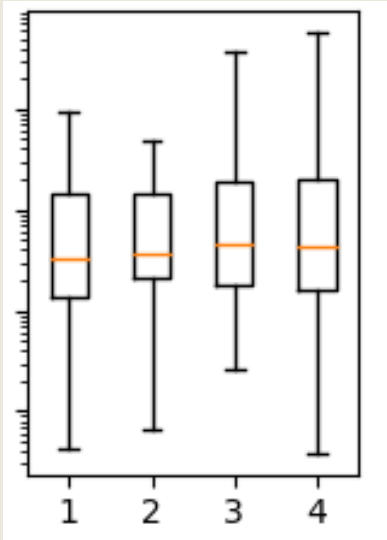


Visual representations: Boxplot

- Boxplot to visualise median and quartiles
  - *Line within box: median*
  - *Box: from Q1 to Q3 (inter-quartile range)*
  - *Whiskers:*
    from min to Q1 and from Q3 to max



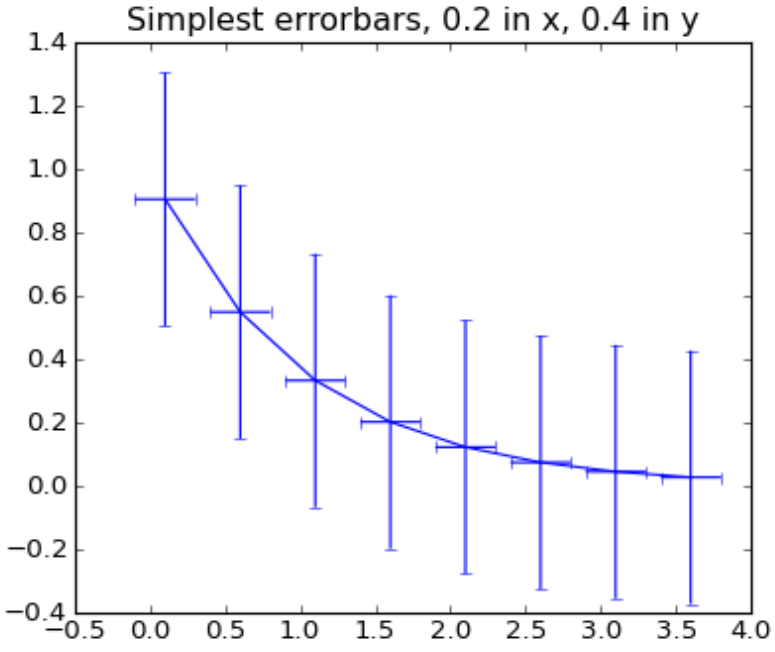
- Can show several side by side for comparison



In Python: matplotlib.pyplot.boxplot
See https://matplotlib.org/examples/pylab_examples/boxplot_demo.html
and https://matplotlib.org/api/_as_gen/matplotlib.pyplot.boxplot.html

Visual representations: error bars

- A graphical representation of the variability of data
- indicate the error or uncertainty in a reported measurement (how precise a measurement is)
- Can use different measurements - e.g. one standard deviation.



In Python: matplotlib.axes.Axes.errorbar
See https://matplotlib.org/1.2.1/examples/pylab_examples/errorbar_demo.html
and https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.errorbar.html

Code profiling: finding bottlenecks in your code

Code profiling: find bottlenecks in your code

- **cProfile** module
- **line_profiler** module for more detail
- Then try to optimize code for these bottlenecks - Tuning

Resources

- The Python Profilers https://docs.python.org/3.7/library/profile.html
- Python 102: How to Profile Your Code https://www.blog.pythonlibrary.org/2014/03/20/python-102-how-to-profile-your-code/
- Profiling Python Like a Boss https://zapier.com/engineering/profiling-python-boss/
- Example on following slides follows https://marcobonzanini.com/2015/01/05/my-python-code-is-slow-tips-for-profiling/

cProfile module

- Standard module, no install required
- Can be called from command line without modifying existing code

```
$ python -m cProfile -o profiling_results profile_test.py
```

Output file (optional)

Code we want to analyse

cProfile example

```
F:\Dropbox\CSN08114 Python>python -m cProfile timing_test1.py
adding up ints from 1 to 1000000
loopy():         0.046799660 (result: 500000500000)
listy():         0.109599352 (result: 500000500000)
pythonic():      0.031199932 (result: 500000500000)
mathematical():  0.000000000 (result: 500000500000)
         23 function calls in 0.185 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.098    0.098 timing_test1.py:12(listy)
        1    0.076    0.076    0.076    0.076 timing_test1.py:13(<listcomp>)
        1    0.000    0.000    0.030    0.030 timing_test1.py:17(pythonic)
        1    0.000    0.000    0.000    0.000 timing_test1.py:21(mathematical)
        1    0.009    0.009    0.185    0.185 timing_test1.py:6(<module>)
        1    0.047    0.047    0.047    0.047 timing_test1.py:6(loopy)
        1    0.000    0.000    0.185    0.185 {built-in method builtins.exec}
        5    0.001    0.000    0.001    0.000 {built-in method builtins.print}
        2    0.052    0.026    0.052    0.026 {built-in method builtins.sum}
        8    0.000    0.000    0.000    0.000 {built-in method time.time}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler'
```

Analysis of cProfile output: pstats module

- Not so easy to eyeball output especially for complex code
- Use **pstats** module to help
  - *Input is file with cProfile results*
  - *Can sort by different columns, e.g. total time*
  - *Can then print e.g. top 10 / top 5 costly*

Analysis of cProfile output: pstats module

```
>>> import pstats
>>> stats = pstats.Stats("profiling_results")
>>> stats.sort_stats("tottime")
<pstats.Stats object at 0x0000000002BDDBA8>
>>> stats.print_stats(5)
Fri Nov 17 14:38:01 2017    profiling_results

         23 function calls in 0.193 seconds

   Ordered by: internal time
   List reduced from 11 to 5 due to restriction <5>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.082    0.082    0.082    0.082 timing_test1.py:13(<listcomp>)
        2    0.054    0.027    0.054    0.027 {built-in method builtins.sum}
        1    0.047    0.047    0.047    0.047 timing_test1.py:6(loopy)
        1    0.009    0.009    0.193    0.193 timing_test1.py:6(<module>)
        5    0.001    0.000    0.001    0.000 {built-in method builtins.print}


<pstats.Stats object at 0x0000000002BDDBA8>
>>>
```

Heuristics for tuning

Remember tuning does not make your program right!

1. Get it right.
2. Test it's right.
3. Profile if slow.
4. Optimise.
5. Repeat from 2.

- (from: https://wiki.python.org/moin/PythonSpeed/PerformanceTips)

How do we know which approach is best?

- Python knowledge - expectations - general heuristics
  - *E.g. list comprehension should be quicker than explicit loop (if we need the list itself)*
  - *E.g. writing to disk takes time*
  - *E.g. printing takes time*
  - *E.g. using imported functions takes time*
- Theoretical approach: calculate O(n) for your algorithm
- time the code execution within Python

Good practice as you go: examples

- Remember strings are immutable - avoid adding to a string (as this will make a copy of the string) - use .join() instead

Do not use the below construct.

```
1 s = ""
2 for x in somelist:
3     s += some_function(x)
```

Instead use this

```
1 slist = [some_function(el) for el in somelist]
2 s = "".join(slist)
```

- Use functions rather than procedural code

```
1 def main():
2     for i in xrange(10**8):
3         pass
4
5 main()
```

is better than

```
1 for i in xrange(10**8):
2     pass
```

Above from http://blog.hackerearth.com/4-Performance-Optimization-Tips-Faster-Python-Code

Good practice as you go: examples

- Optimise/avoid loops
  - *Replace with list comprehension or other iterable methods*
  - *if you calculate values that don't change in a loop, move them outside the loop*
- Avoid dots (function references)
- Import statements have overheads - do not import them inside a function
- Use local variables

Above from https://wiki.python.org/moin/PythonSpeed/PerformanceTips

More about optimising loops

- Avoid the use of dots within a loop
  - *E.g. every time str.upper() is called, python evaluates the method*
  - *So do this in a variable before the loop rather than in the loop*
  - *Optimise_loop.py example on following slides*

From: https://dzone.com/articles/6-python-performance-tips

Optimise loop example

Avoid the use of dots within a loop

```
#loop optimisation by moving dots to variables
# modified from https://dzone.com/articles/6-python-performance-tips

# function1 should be faster than function2
# as it evaluates str.upper and upperlist.append methods only once
def function1(lowerlist, upperlist=[]):
    upper = str.upper
    append = upperlist.append
    for word in lowerlist:
        append(upper(word))
    return(upperlist)


#function2 should be slower as it evaluates methods in every iteration
def function2(lowerlist, upperlist=[]):
    for word in lowerlist:
        upperlist.append(str.upper(word))
    return(upperlist)


# now run the functions for comparison
lowerlist = ['this', 'is', 'lowercase']*1000000
upperlist1 = function1(lowerlist,[])
upperlist2 = function2(lowerlist,[])
```

Optimise loop example

Avoid the use of dots within a loop

```
C:\Users\Petra\Dropbox\CSN08114 Python>python -m cProfile optimise_loop.py
         6000005 function calls in 3.434 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    1.417    1.417    1.922    1.922 optimise_loop.py:15(function2)
        1    0.018    0.018    3.434    3.434 optimise_loop.py:6(<module>)
        1    1.025    1.025    1.493    1.493 optimise_loop.py:6(function1)
        1    0.000    0.000    3.434    3.434 {built-in method builtins.exec}
  6000000    0.973    0.000    0.973    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

So here function1 is almost 25% faster than function2

Q: How can we do even better than this???

Optimise loop example

Avoid the use of dots within a loop

```
#function3 uses list comprehension instead of a loop
def function3(lowerlist):
    upper = str.upper
    upperlist = [upper(word) for word in lowerlist]
    return(upperlist)
```

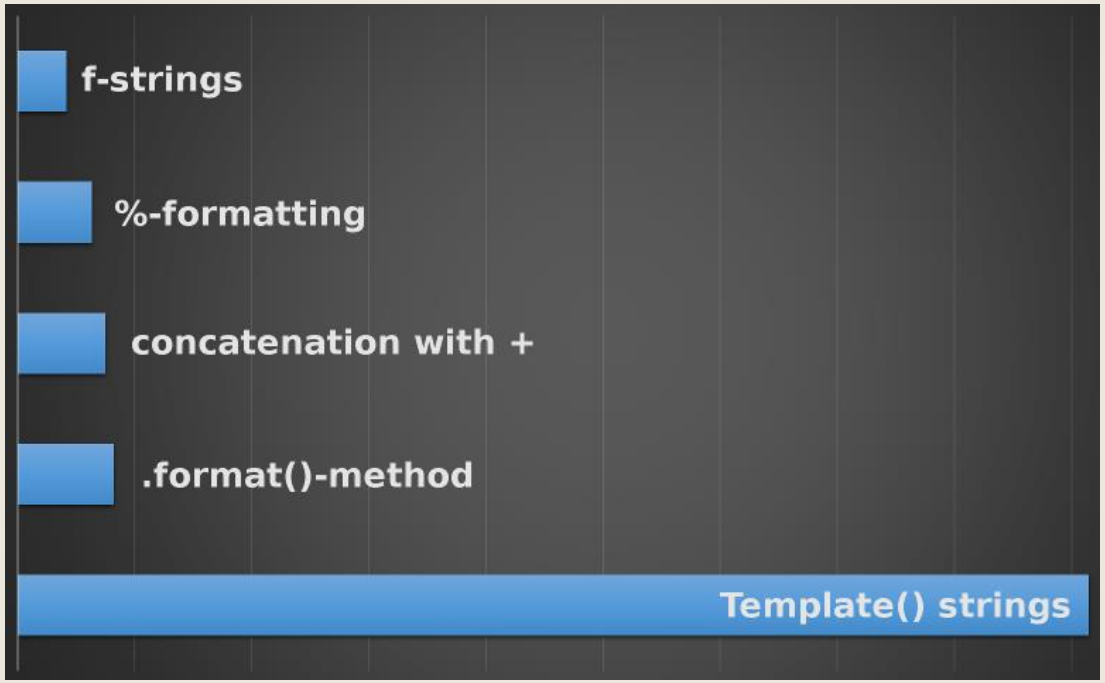
```
C:\Users\Petra\Dropbox\CSN08114 Python>python -m cProfile optimise_loop.py
all results are identical
         6000008 function calls in 4.647 seconds
```

function3 is almost 50% faster than function1

```
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    1.448    1.448    1.927    1.927 optimise_loop.py:15(function2)
        1    0.000    0.000    0.869    0.869 optimise_loop.py:21(function3)
        1    0.869    0.869    0.869    0.869 optimise_loop.py:23(<listcomp>)
        1    0.347    0.347    4.647    4.647 optimise_loop.py:6(<module>)
        1    1.033    1.033    1.502    1.502 optimise_loop.py:6(function1)
        1    0.000    0.000    4.647    4.647 {built-in method builtins.exec}
        1    0.001    0.001    0.001    0.001 {built-in method builtins.print}
  6000000    0.947    0.000    0.947    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

How best to format strings e.g. for print?

- F-strings are fast!
- details (and below diagram) at https://cito.github.io/blog/f-strings/



Practical Lab 06

Some Resources

Here are all the links given previously in a neat summary. The lab exercises will give links that are specifically useful for each exercise.

- Using yield instead of return https://www.pythoncentral.io/python-generators-and-yield-keyword/
- Online calculators https://www.calculatorsoup.com/calculators/discretemathematics/permutations.php , https://www.calculatorsoup.com/calculators/discretemathematics/combinations.php
- Animation about traveling salesman problem and solutions https://www.youtube.com/watch?v=SC5CX8drAtU
- itertools tutorial https://www.geeksforgeeks.org/permutation-and-combination-in-python/
- scipy functions https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.perm.html, https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.comb.html
- Mean, median and mode http://www.abs.gov.au/websitedbs/a3121120.nsf/home/statistical+language+-+measures+of+central+tendency, https://learnandteachstatistics.wordpress.com/2013/04/29/median/
- Measures of spread http://www.abs.gov.au/websitedbs/a3121120.nsf/home/statistical+language+-+measures+of+spread

Links last checked: 17/10/18

Resources (continued)

Here are all the links given previously in a neat summary. The lab exercises will give links that are specifically useful for each exercise.

- matplotlib.pyplot.boxplot https://matplotlib.org/examples/pylab_examples/boxplot_demo.html , https://matplotlib.org/api/_as_gen/matplotlib.pyplot.boxplot.html
- matplotlib.axes.Axes.errorbar https://matplotlib.org/1.2.1/examples/pylab_examples/errorbar_demo.html , https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.errorbar.html
- The Python Profilers https://docs.python.org/3.7/library/profile.html
- Profiling and Tuning tips and examples https://www.blog.pythonlibrary.org/2014/03/20/python-102-how-to-profile-your-code/, https://zapier.com/engineering/profiling-python-boss/, https://marcobonzanini.com/2015/01/05/my-python-code-is-slow-tips-for-profiling/ , https://wiki.python.org/moin/PythonSpeed/PerformanceTips, http://blog.hackerearth.com/4-Performance-Optimization-Tips-Faster-Python-Code , https://dzone.com/articles/6-python-performance-tips
- F-strings https://cito.github.io/blog/f-strings/